
Article:
Zakharov, Yuriy orcid.org/0000-0002-2193-4334, Nascimento, Vitor, Caiado De Lamare, Rodrigo et al. (1 more author) (2017) Low-complexity DCD-based sparse recovery algorithms. IEEE Access. 7950926. 12737 - 12750. ISSN 2169-3536

https://doi.org/10.1109/ACCESS.2017.2715882

Low-complexity DCD-based sparse recovery algorithms

Yuriy Zakharov[†], *Senior Member*, *IEEE*, Vítor Nascimento[††], *Senior Member*, *IEEE*, Rodrigo de Lamare[†‡], *Senior Member*, *IEEE*, and Fernando Goncalves de Almeida Neto[⋆]

***Abstract*— Sparse recovery techniques find applications in many areas. Real-time implementation of such techniques has been recently an important area for research. In this paper, we propose computationally efficient techniques based on dichotomous coordinate descent (DCD) iterations for recovery of sparse complex-valued signals. We first consider $\ell_2\ell_1$ optimization that can incorporate *a priori* information on the solution in the form of a weight vector. We propose a DCD-based algorithm for $\ell_2\ell_1$ optimization with a fixed ℓ_1-regularization, and then efficiently incorporate it in reweighting iterations using a *warm start* at each iteration. We then exploit homotopy by sampling the regularization parameter and arrive at an algorithm that, in each homotopy iteration, performs the $\ell_2\ell_1$ optimization on the current support with a fixed regularization parameter and then updates the support by adding/removing elements. We propose efficient rules for adding and removing the elements. The performance of the homotopy algorithm is further improved with the reweighting. We then propose an algorithm for $\ell_2\ell_0$ optimization that exploits homotopy for the ℓ_0 regularization; it alternates between the least-squares (LS) optimization on the support and the support update, for which we also propose an efficient rule. The algorithm complexity is reduced when DCD iterations with a *warm start* are used for the LS optimization, and, as most of the DCD operations are additions and bit-shifts, it is especially suited to real time implementation. The proposed algorithms are investigated in channel estimation scenarios and compared with known sparse recovery techniques such as the matching pursuit (MP) and YALL1 algorithms. The numerical examples show that the proposed techniques achieve a mean-squared error smaller than that of the YALL1 algorithm and complexity comparable to that of the MP algorithm.**

***Index Terms*— coordinate descent, DCD, homotopy, sparse recovery**

I. Introduction

Sparse recovery techniques find applications in many areas, including channel estimation [1]–[8], array beamforming [9]–[12], adaptive filtering [13]–[17], and many others that require low-complexity algorithms suitable for real-time implementation. Real-time implementation of sparse recovery techniques, particularly on Field Programmable Gate Arrays (FPGAs), has been recently an important area for research [18]–[25]. There are two families of techniques for finding sparse representations: convex optimization and greedy methods. Generally, greedy techniques have lower complexity and require lower numerical precision [22]. Therefore, when it comes to hardware (such as FPGA) implementation, the matching pursuit (MP) algorithm [18], [20], [22] or other greedy algorithms such as CoSaMP [21], [26] or OMP [23], [27] are considered as the most suitable candidates. The MP is a popular greedy technique that possesses especially low complexity and therefore finds many practical applications [3], [13], [22], [28]–[30]. However, in the accuracy performance, it is inferior to many other techniques.

In the family of convex optimization techniques, there are many algorithms possessing high performance, and recently the YALL1 algorithm implementing the alternating direction method [31], has become popular due to its high accuracy [5], [10]. We will consider the MP complexity and the YALL1 recovery performance as benchmarks when analyzing the algorithms that we propose in this paper. The MP, YALL1 as well as our algorithms can deal with complex-valued problems. Often, applications such as channel estimation, beamforming, equalization and many others also require solving complex-valued problems [7], [11], [32]. The family of algorithms capable of dealing with complex-valued problems is scarcer than the family of algorithms for real-valued problems. Importantly, it is not always straightforward to transform real-valued algorithms into complex-valued counterparts [7], [33]–[35]. A complex-valued algorithm can exploit the coupling (common support) of the real and imaginary parts of a signal [35] and potential non circularity of a signal, whereas the real-valued counterpart does not have such a feature. Here we will focus on sparse recovery algorithms for complex-valued problems.

It has been previously recognized that the coordinate descent (CD) search has an inherent property of being low complexity when signals are sparse [36]–[40]. We derive our algorithms applying CD iterations for solving $\ell_2\ell_1$ and $\ell_2\ell_0$ optimization problems. Specifically, we exploit dichotomous CD (DCD) iterations [36] that minimize the use of multiplications, thus resulting in algorithms especially well suited to real-time implementation, e.g. using FPGAs [41]–[44].

In the family of greedy techniques, algorithms based on ℓ_1-homotopy [45] demonstrate high accuracy for recovering sparse signals and are of lower complexity in comparison with many other techniques [7], [46], [47]. It is important to note that even if the homotopy approach requires solving a sequence of optimization problems for a corresponding sequence of values of a regularization parameter, due to the

[†]Department of Electronic Engineering, University of York, UK, yury.zakharov@york.ac.uk, rodrigo.delamare@york.ac.uk. [††]Dept. of Electronic Systems Eng., University of São Paulo, Brazil, vitor@lps.usp.br. [‡]CETUC, PUC-Rio, Rio de Janeiro, Brazil, delamare@cetuc.puc-rio.br. [⋆]Federal Rural University of Pernambuco, Brazil, fernando.galmeida@ufrpe.br. The work of Y. V. Zakharov was partly supported by FAPESP-York-2015 grant. The work of V. H. Nascimento was partly supported by CNPq 0306268/2014-0 and FAPESP 2014/50765-6 grants. The work of R. de Lamare was partly supported by FAPESP-York-2015, FAPERJ, and CNPq grants. A part of the material on the $\ell_2\ell_1$ optimization was presented in the conference Asilomar-2012.

iterative nature of the DCD algorithm, if we use the current estimate as the initial condition for the next homotopy iteration (*warm* initialization), the complexity is in fact reduced, since only a few updates are necessary at each homotopy iteration. We will be using this approach to derive our algorithms.

A priori information on the support of sparse signals can significantly improve the algorithm performance. In the extreme case, when the support is perfectly known, the performance of the *oracle* algorithm is achieved. We will use the *oracle* performance as another benchmark for comparison with the performance of the proposed algorithms. However, such a knowledge of the support is most often unavailable. There have previously been proposed techniques for estimating and further refining the information on the support in a set of reweightening iterations and incorporating these estimates in the cost function [48]–[51]. We will exploit this idea to improve the recovery performance of the proposed algorithms.

In this paper, we first derive DCD-based algorithms for the weighted ℓ_1 penalized regression. The ℓ_1-DCD algorithm updates all elements of the unknown vector. As the vector length can be high, the complexity can also be high. The alternative is a greedy-like algorithm that only deals with a relatively small number of elements within the currently identified support. Therefore, we develop a greedy algorithm that is based on homotopy with respect to the ℓ_1 regularization, and the ℓ_1-DCD iterations are used for updating the solution within the support. We formulate optimal rules for adding and removing elements from the support. The performance of the homotopy algorithm is further improved with the reweighting. Note that after publishing the initial version of the $\ell_2\ell_1$ homotopy algorithm in the Asilomar conference [52], it was further developed in [53]–[55] for application in radar imaging and in [56] for application in adaptive filtering.

We then propose an algorithm for $\ell_2\ell_0$ minimization that exploits homotopy for the ℓ_0 regularization (denoted as Hℓ_0 direct-LS algorithm). It alternates between the least-squares (LS) optimization on the support and the support update, for which we also propose efficient rules. We then apply the DCD iterations to solve the sequence of LS problems and arrive at a computationally efficient (Hℓ_0-DCD) algorithm.

The paper is organized as follows. Section II describes the signal model. In Section III, we derive the ℓ_1-DCD algorithm for $\ell_2\ell_1$ minimization. In Section IV, we present the homotopy DCD (Hℓ_1-DCD) algorithm. Section V presents homotopy algorithms for $\ell_2\ell_0$ minimization: Hℓ_0 direct-LS and Hℓ_0-DCD algorithms. Section VI contains numerical examples and, finally, Section VII presents conclusions.

Notation: We use capital and small bold fonts to denote matrices and vectors, respectively; e.g. $\mathbf{A}$ is a matrix and $\mathbf{x}$ a vector. Elements of the matrix and vector are denoted as $A_{n,p}$ and x_n, respectively. We use I and I^c to denote a support (indexes of non-zero elements) and its complement, respectively; the cardinality of I is denoted as $|I|$. We also denote: $\mathbf{A}^{(q)}$ the qth column of $\mathbf{A}$; $\mathbf{A}^H$ the Hermitian transpose of $\mathbf{A}$; trace$[\mathbf{R}]$ the trace of $\mathbf{R}$; $\mathbf{A}_I$ the matrix obtained from $\mathbf{A}$ keeping only columns corresponding to support I; $\mathbf{R}_{I,I}$ the $|I|\times|I|$ matrix obtained from $\mathbf{R}$ extracting elements from rows and columns with indexes in the support I; $\mathbf{x}_I$ the subset of $\mathbf{x}$ that contains non-zero entries from $\mathbf{x}$ corresponding to the support I; $\Re\{\cdot\}$ and $\Im\{\cdot\}$ are the real and imaginary parts of a complex number, respectively.

II. Signal model

We consider the linear model

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{n} \tag{1}$$

where $\mathbf{A} \in \mathbb{C}^{M\times N}$ is the observation matrix, $\mathbf{n} \in \mathbb{C}^{M\times 1}$ the noise vector, $\mathbf{y} \in \mathbb{C}^{M\times 1}$ the observed signal, and $\mathbf{x} \in \mathbb{C}^{N\times 1}$ the unknown signal. We are especially interested in the case, in which $M < N$ and all variables in (1) are complex-valued. We also assume that only $K < M$ elements of vector $\mathbf{x}$ are non-zero, i.e. the vector $\mathbf{x}$ is sparse, and the support (index set of non-zeros) is unknown.

Applications of sparse recovery algorithms differ in the possibility of precomputing the matrix $\mathbf{R} = \mathbf{A}^H\mathbf{A}$. If $\mathbf{R}$ cannot be precomputed, the complexity of algorithms may be dominated by the on-line computation of $\mathbf{R}$ or its submatrices. In other applications, $\mathbf{R}$ can be precomputed or updated in real-time with low complexity, e.g. as in adaptive filtering [3], [14], [41], [57], channel estimation [22], beamforming [9]–[11], etc. Here we are interested in applications where $\mathbf{R}$ is available.

III. DCD algorithm for $\ell_2\ell_1$ optimization with fixed regularization

We consider the minimization of the $\ell_2\ell_1$ cost function

$$J_{\mathbf{w},\tau}(\mathbf{x}) = \frac{1}{2}||\mathbf{y} - \mathbf{A}\mathbf{x}||_2^2 + \tau\mathbf{w}^T|\mathbf{x}| \tag{2}$$

with a fixed regularization parameter τ, where $|\mathbf{x}|$ is a vector of element-wise magnitudes of entries in $\mathbf{x}$, and $\mathbf{w}$ is a weight vector. We now present a direct (non-homotopy) DCD approach for minimization of the cost function in (2), starting with the weight selection and support estimation.

If $w_k = 1$ for $k = 1, \ldots, N$, this problem is also known as basis pursuit denoising (BPDN) [58]. We use as benchmark in our simulations an *oracle* algorithm, which knows the true location of the nonzero entries. The *oracle* algorithm can be obtained by choosing $w_k = 0$ for k within the support, $w_k = 1$ otherwise, and with τ very large. See other examples of the weight vector in [40], [48], [50].

The weight vector can also be updated in N_s reweighting iterations and in each iteration, a new problem (2) is solved [7], [49]. We will be using the following updating mechanism. In every reweighting iteration with index $s = 1, \ldots, N_s$, a weight support Γ_w is identified using hard thresholding:

$$\Gamma_w = \left\{k:\ |x_k| > (\mu_w)^s T_w \max_n\{|x_n|\}\right\} \tag{3}$$

where $\mu_w \in [0, 1]$ and T_w are adjusted parameters. Within the support Γ_w, the weight entries are set to a value β^s, where $\beta \in [0, 1)$ is an adjusted parameter. In the case of $\beta = 0$ and $\mu_w = 1$, we arrive at the weights in [49]. In the case of $T_w = 1$, $\beta = 0$ and $\mu_w = 1/2$, we obtain the weights in [59]. The threshold is reduced with every reweighting iteration and (if $\beta > 0$) the weights are also reduced. This is reasonable

TABLE I

GENERAL STRUCTURE OF SPARSE RECOVERY ALGORITHMS BASED ON $\ell_2\ell_1$ OPTIMIZATION WITH REWEIGHTING ITERATIONS

	Initialization: $\mathbf{c} = \mathbf{b} = \mathbf{A}^H\mathbf{y}$, $\mathbf{x} = \mathbf{0}$, $\mathbf{w} = \mathbf{1}_N$, $\mathbf{R} = \mathbf{A}^H\mathbf{A}$ $C_u = 0$, $C_i = 0$, $T_c = \mu_c \max_k \lvert c_k\rvert^2$
1:	**for** $s = 1, \ldots, N_s$ **do**
2:	Solve (2) using ℓ_1-DCD or Hℓ_1-DCD algorithm
3:	For algorithms using i DCD iterations, of which u iterations are *successful*, compute: $C_u \leftarrow C_u + u$, $C_i \leftarrow C_i + i$ **if** $C_u \geq N_u$ **then** the algorithm stops (C_u and C_i are used in the analysis of the algorithm complexity)
4:	Update the weight vector $\mathbf{w}$ on the support Γ_w using (3)
5:	Debiasing according to (5)

since, in general, the estimation accuracy improves with every reweighting iteration. The term $(\mu_w)^s$ provides an exponential reduction of the threshold, while β^s provides an exponential reduction of the weights. The parameter μ_w defines the rate of reducing the threshold, whereas the product $\mu_w T_w$ defines the initial threshold. Note that by choosing $\beta = 1/2$ and all initial weights equal to one, the application of the weights does not require multiplications, but only bit-shifts (that are much less costly to implement in hardware).

We are interested in applications where the objective is a low mean squared error (MSE) on the estimated support, e.g. such as estimation of sparse multipath communication channels [1]–[4], [7]. Therefore, after a sparse recovery algorithm identifies the support and terminates, a debiasing on the support should be done. The final support I can be the output of the sparse recovery algorithm itself or it can be identified, e.g. using hard thresholding, as a set of elements in the solution $\mathbf{x}$, satisfying the condition

$$I = \left\{k \ : \ |x_k| > \mu_{\text{d}} \max_n\{|x_n|\}\right\} \tag{4}$$

where $\mu_{\text{d}} \in [0, 1)$ is a predefined parameter. Using the finally estimated support I, the debiasing stage computes the minimum MSE (MMSE) estimate of the vector $\mathbf{x}$ as the solution to the equation:

$$(\mathbf{R}_{I,I} + \nu\mathbf{I}_{|I|})\mathbf{x}_I = \mathbf{A}_I^H\mathbf{y}, \tag{5}$$

where $\nu = \sigma^2|I|/\text{trace}[\mathbf{R}_{I,I}]$, σ^2 is the noise variance and $\mathbf{I}_{|I|}$ is the $|I| \times |I|$ identity matrix.

The general structure of sparse recovery algorithms proposed in this paper is shown in Table I. For minimizing the cost function (step 2) at each reweighting iteration s, i DCD iterations are used among which there are u *successful* iterations (that is, u iterations in which the solution vector is updated, see [36], [42], [60]; note that the term *successful* iterations was introduced in [60]). We set an upper limit N_u to the total number C_u of successful iterations (considering all reweighting iterations); the algorithm stops as soon as $C_u = N_u$. The parameter T_c is used as a threshold to stop the minimization at step 2; this is checked within the algorithm performed at step 2 (see step 10 in Table II). The total number of DCD iterations C_i and the total number of *successful* DCD iterations C_u determine the algorithm complexity, as detailed later in this section.

The function in (2) is convex and we use DCD iterations to minimize it. One difference between our DCD-based algorithms and previously proposed CD algorithms [37]–[39], [61] is that we do not optimize the step-size for every CD iteration, but instead we use a set of step-sizes defined by the fixed-point representation of the solution. Thus, our CD search is an inexact line search [62], [63] as opposed to the exact line search in most previously proposed CD algorithms. Although, for a particular iteration, the exact line search achieves a higher reduction of the cost function, with an inexact line search the convergence to the true solution in a sequence of iterations can be even faster. In [41] and [42], this was demonstrated when comparing the exact CD search and the DCD search for solving the LS problem. Importantly, the DCD search allows a large reduction in the number of operations; in many cases, the algorithm complexity is dominated by the complexity of *successful* iterations, which typically represent a small part of all DCD iterations, especially for sparse solutions [36]. Moreover, most of the operations are additions and bit-shifts which, together with the fixed-point representation of the solution, make DCD iterations attractive for implementation on real-time design platforms, such as FPGAs [42].

Consider DCD iterations for minimizing the cost function (2). At every iteration, only the p-th element of the solution vector $\mathbf{x}$ may be updated as $\tilde{\mathbf{x}} = \mathbf{x} + \alpha\mathbf{e}_p$, where α is a complex-valued scalar and the vector $\mathbf{e}_p$ has the p-th element equal to one and the others zero. The update should only be done if the cost function is reduced, i.e. if

$$\Delta J = J_{\mathbf{w},\tau}(\mathbf{x} + \alpha\mathbf{e}_p) - J_{\mathbf{w},\tau}(\mathbf{x}) < 0$$

This can be written in the form

$$\begin{aligned}\Delta J = \frac{1}{2}|\alpha|^2 R_{p,p} \quad &- \quad \Re\{\alpha^*(b_p - \mathbf{R}^{(p)H}\mathbf{x})\} \\ &+ \quad \tau w_p(|x_p + \alpha| - |x_p|)\end{aligned} \tag{6}$$

where $\mathbf{b} = \mathbf{A}^H\mathbf{y}$. Defining the residual vector $\mathbf{c} = \mathbf{b} - \mathbf{R}\mathbf{x}$, we obtain

$$\Delta J = \frac{1}{2}|\alpha|^2 R_{p,p} - \Re\{\alpha^* c_p\} + \tau w_p(|x_p + \alpha| - |x_p|) \tag{7}$$

Starting from $\mathbf{x} = \mathbf{0}$ and $\mathbf{c} = \mathbf{b}$, the residual vector can be recursively updated at every coordinate descent iteration as $\mathbf{c} \leftarrow \mathbf{c} - \alpha\mathbf{R}^{(p)}$. If α is a power-of-two number, the update of $\mathbf{c}$ requires M complex additions and no multiplications. Note also that if $\Delta J \geq 0$, then no update is necessary - we call this low-cost case an *unsuccessful* iteration. If $\Delta J < 0$, an update is necessary - this is a *successful* iteration.

When using DCD iterations, it is assumed that elements of the solution vector have a fixed-point representation with M_b bits within an amplitude interval $[-H, H]$. The choice of H may be defined from the maximum magnitude of elements in $\mathbf{x}$. Note that the choice of H is not unique for DCD; in any system with fixed-point representation of signals, one has to decide on the maximum possible amplitude of the signals. It is preferable to choose H as the smallest power-of-two number satisfying $H \geq \max_q\{|\Re[x_q]|, |\Im[x_q]|\}$. However, the choice is not very critical; see discussion on the choice of H in [42]. The DCD iterations start updates from the most significant bits

TABLE II
ℓ_1-DCD ALGORITHM

	Input: $\mathbf{c}$, $\mathbf{R}$, $\mathbf{x}$, H, N_u, M_b, T_c
	Initialization: $\delta = H$, $u = 0$, $i = 0$
1:	**for** $m = 1, \ldots, M_b$ **do**
2:	$\delta = \delta/2$, $\boldsymbol{\alpha} = [\delta,\ -\delta,\ j\delta, -j\delta]$, Flag = 0
3:	**for** $p = 1, \ldots, N$ **do**
4:	**for** $k = 1, \ldots, 4$ **do**
5:	$\Delta J = \frac{1}{2}\delta^2 R_{p,p} - \Re\{\alpha_k^* c_p\} + \tau w_p(\lvert x_p + \alpha_k\rvert - \lvert x_p\rvert)$ $i \leftarrow i + 1$
6:	**if** $\Delta J < 0$ **then**
7:	$x_p \leftarrow x_p + \alpha_k$, $\mathbf{c} \leftarrow \mathbf{c} - \alpha_k \mathbf{R}^{(p)}$
8:	Flag = 1, $u \leftarrow u + 1$
	if $u = N_u$ **then** the algorithm stops
9:	**if** Flag = 1, go to step 3
10:	**if** $\max_k \lvert c_k\rvert^2 < T_c$ **then** the algorithm stops
	Output: $\mathbf{c}$, $\mathbf{x}$, u, i

of coordinates towards less significant bits. This is controlled by a step-size $\delta > 0$ that starts with $\delta = H$ and is reduced as $\delta \leftarrow \delta/2$ for less significant bits.

In a DCD algorithm for complex-valued problems, for every coordinate, there are four possible directions on the complex plane for updating: 1, -1, j and $-j$, where $j = \sqrt{-1}$. Consequently, there are four values by which a coordinate can be updated. These are the scalar values α_k, $k = 1, \ldots, 4$, defined by these four directions and the step-size δ as elements of the vector $\boldsymbol{\alpha} = [\delta,\ -\delta,\ j\delta,\ -j\delta]$.

There can be different strategies for selecting coordinates for updates. The most often used are cyclic and leading [41] (also called greedy [61]) selections. Leading CD iterations require costly computations for selection of the best coordinate for updating. With cyclic DCD iterations, we do not need to find the minimum of the cost function over all possible updates. The cyclic DCD algorithm for minimizing $J_{\mathbf{w},\tau}$ is shown in Table II. [We have also developed a leading DCD algorithm for minimizing the cost function in (2) that provides an accuracy similar to that of the cyclic DCD algorithm. However, its complexity is somewhat higher than the complexity of the cyclic DCD algorithm. Therefore, here, we only present the cyclic DCD algorithm.] In addition to the parameters of the algorithm described above (M_b and H), we also introduce the maximum number of *successful* iterations (i.e. iterations where the solution is updated) N_u and a parameter μ_c that defines the threshold T_c for residual magnitudes; $\mu_c \in [0, 1)$ is a predefined parameter (see Table I). If all the magnitudes are smaller than the threshold, the algorithm stops.

The complexity of the cyclic DCD (we call it ℓ_1-DCD) algorithm in (real-valued) flops is given by

$$\begin{aligned} P_{\ell_1\text{-DCD}} &\approx 8MN + 4N + 11C_i + 2NC_u \\ &+ 4NM_b + 2N_{\text{deb}}L_g \end{aligned} \tag{8}$$

The first and second terms in (8) are for computing $\mathbf{c}$ and T_c at the initialization stage (see Table I); these operations are multiplications and additions. The term $11C_i$ is the complexity of C_i tests at steps 5 and 6 in Table II; each computation involves analysis of one direction on the complex plane for one element, which can be done with 11 real-valued operations, including multiplications, additions, and square-roots (required to compute the absolute values $|x_p|$ and $|x_p + \alpha_k|$). The exact value of C_i, the number of times the test at step 6 is evaluated, is difficult to predict; $C_i \geq N_u$ and tends to increase as the vector length N increases. The term $2NC_u$ is for updating $\mathbf{c}$ in the overall C_u *successful* (i.e. when $\Delta J < 0$) DCD iterations (step 7); each update requires only $2N$ real-valued additions, as multiplications by α_k are bit-shift operations. The next term in (8), $4NM_b$, is the complexity of computing square magnitudes $|c_k|^2$ of the residual vector and search for the maximum to check the termination condition at step 10; this is done for every bit m, thus the factor M_b, and involves multiplications and additions. The debiasing (last term) can be efficiently done by using extra N_{deb} DCD iterations at the finally fixed support of cardinality L_g using the previously found estimate of $\mathbf{x}$ as a *warm start*. The DCD iterations for debiasing are similar to that in Table VI below and only involve additions.

The complexity for the ℓ_1-DCD algorithm in (8) and other algorithms below are given in terms of flops. We are using this measure because of the difficulty in estimating complexity for the YALL1 algorithm (which we use here as a benchmark) and the proposed ℓ_0 direct-LS algorithm, and because the true complexity will depend on the particular hardware implementation (e.g., DSPs, which have one-cycle multiplication units, or FPGAs). This way of measuring complexity tends to overestimate the complexity for DCD-based algorithms in FPGA implementations, because DCD avoids multiplications and divisions, which are complicated to implement in hardware [64].

Although the ℓ_1-DCD algorithm demonstrates a high recovery performance and relatively low complexity (as will be seen in Section VI), the complexity can be further reduced. Note that for high N, C_i can be high, and in this case the main contribution to the ℓ_1-DCD complexity are computations at step 5 in Table II, which are repeated C_i times. Moreover, these (and only these) computations involve the square-root operation. Although such an operation can be efficiently implemented using DCD iterations [65], it is still more complicated for hardware implementation than addition and multiplication. The number of these computations can be significantly reduced if they are only performed on the currently identified support with a size $|I|$ that is typically significantly lower than N. This can be done using the homotopy approach as described in the next section.

IV. HOMOTOPY DCD ALGORITHM FOR $\ell_2\ell_1$ OPTIMIZATION

For minimization of $J_{\mathbf{w},\tau}$ in (2) with further reduced complexity, we now use homotopy with respect to the regularization parameter τ. If τ is high, the second term in (2) dominates the cost function and forces the cardinality of the presumed support to zero. The strategy is to select the minimum possible value ($\tau_{\max}$) for the regularization parameter τ, for which the solution is still all-zeros, and, in homotopy iterations, generate a decreasing sequence of values of τ using uniform sampling in the log scale; this is similar to the strategy in [40]. We will start with $\tau = \tau_{\max}$ and empty support $I = \emptyset$. For every

τ, we will update the support and, on the updated support, minimize the cost function $J_{\mathbf{w},\tau}$. The algorithm will terminate if $\tau \leq \tau_{\min}$, where $\tau_{\min} = \mu_\tau \tau_{\max}$ and $\mu_\tau \in [0,1]$ is a predefined parameter, or if another termination condition is met. Starting with zero support enables us to work always with a small support, keeping the dimension of the problem at each step low and significantly reducing the complexity.

We need to find rules for adding and removing elements into/from the support.

Proposition 1: Let I be the support at some homotopy iteration and the current solution to the $\ell_2\ell_1$ minimization be $\mathbf{x}$. Let $\mathbf{b} = \mathbf{A}^H\mathbf{y}$, $\mathbf{R} = \mathbf{A}^H\mathbf{A}$, and $\mathbf{c} = \mathbf{b} - \mathbf{R}\mathbf{x}$. Adding the t-th element, $t \in I^c$, into the support according to the rule

$$t = \arg\max_{k \in I^c} \frac{(|c_k| - \tau w_k)^2}{R_{k,k}} \quad \text{s.t.} \quad |c_t| > \tau w_t \tag{9}$$

leads to reduction of the cost function (2) for a properly chosen x_t. The value

$$x_t = \frac{c_t}{R_{t,t}} \left(1 - \frac{\tau w_t}{|c_t|}\right) \tag{10}$$

minimizes the cost function.

Proof. To prove this proposition, we analyze the change of the cost function due to the update $\tilde{\mathbf{x}} = \mathbf{x} + \alpha \mathbf{e}_t$, where $t \in I^c$, i.e. $x_t = 0$. The cost function will be changed by $\Delta J = J_{\mathbf{w},\tau}(\mathbf{x} + \alpha \mathbf{e}_t) - J_{\mathbf{w},\tau}(\mathbf{x})$. We need to check if an α exists that results in $\Delta J < 0$. We rewrite ΔJ as

$$\begin{aligned} \Delta J &= \frac{1}{2}|\alpha|^2 R_{t,t} - \Re\{\alpha^* c_t\} + \tau|\alpha| w_t \\ &= \frac{1}{2}|\alpha|^2 R_{t,t} - |\alpha| \Re\{e^{-j\arg(\alpha)} c_t\} + \tau|\alpha| w_t \end{aligned}$$

It is seen that a minimum of ΔJ over $\arg(\alpha)$ is achieved if $\arg(\alpha) = \arg(c_t)$ and, in this case, we have

$$\Delta J = \frac{1}{2}|\alpha|^2 R_{t,t} - |\alpha||c_t| + \tau|\alpha| w_t \tag{11}$$

To find $|\alpha|$ minimizing (11), we solve

$$\frac{\partial \Delta J}{\partial |\alpha|} = |\alpha| R_{t,t} - |c_t| + \tau w_t = 0 \tag{12}$$

and obtain that the minimum of ΔJ over $|\alpha|$ is achieved at $|\alpha| = (|c_t| - \tau w_t)/R_{t,t} > 0$. Thus, ΔJ is minimized and negative if $|c_t| > \tau w_t$ and

$$\alpha = \frac{1}{R_{t,t}}(|c_t| - \tau w_t) e^{j\arg(c_t)} = \frac{c_t}{R_{t,t}} \left(1 - \frac{\tau w_t}{|c_t|}\right)$$

In this case, the largest decrement of the cost function $J_{\mathbf{w},\tau}$ is given by $\Delta J = -(1/2)(|c_t| - \tau w_t)^2 / R_{t,t}$. Thus, the rule for adding a new element into the support can be formulated as in (9) and (10). □

Note that we are computing the minimum here just to show that adding a new element to the support would lead to a reduction of the cost function. In order to keep computational complexity low, and to keep the fixed-point structure of the solution, we do not actually update the solution using the optimal value of x_t in (10); see Table III and the discussion further below.

This proposition also determines the starting value $\tau_{\max}$ of the regularization parameter τ. If $I = \emptyset$, the first element to be added into the support should be the one that maximizes $(|b_t| - \tau w_t)^2 / R_{t,t}$ over $t = 1, \ldots, N$. If $w_t = \text{const} > 0$ and $R_{t,t} = 1$, we arrive at the rule: $t = \arg\max_k |b_k|^2$. In the general case, $\tau_{\max}$ is chosen according to:

$$\tau_{\max} = \max_{k \in \Gamma_w} \frac{|b_k|}{w_k}, \quad \text{where} \quad \Gamma_w = \{k : w_k > 0\} \tag{13}$$

TABLE III
Hℓ_1-DCD ALGORITHM

Initialization: $\mathbf{x} = \mathbf{0}$, $I = \emptyset$, $\mathbf{c} = \mathbf{b} = \mathbf{A}^H\mathbf{y}$, $\mathbf{R} = \mathbf{A}^H\mathbf{A}$
Choose the first element t into the support: $I = \{t\}$; compute $\tau_{\max}$
Repeat until a termination condition is met:
  Solve $\min_{\mathbf{x}_I} ||\mathbf{y} - \mathbf{A}_I \mathbf{x}_I||_2^2 + \tau \mathbf{w}_I^T |\mathbf{x}_I|$ on the support I and update $\mathbf{c}$ using ℓ_1-DCD iterations
  Update the regularization parameter: $\tau \leftarrow \gamma\tau$
  Remove an index t from the support I according to rule (14)
  If the tth element is removed **then** $\mathbf{c} \leftarrow \mathbf{c} + x_t \mathbf{R}^{(t)}$
  Add an index t into the support I according to rule (9), but keep the corresponding element x_t equal to zero

Proposition 2: Let I be the support at some homotopy iteration and the current solution to the $\ell_2\ell_1$ minimization be $\mathbf{x}$. Let $\mathbf{b} = \mathbf{A}^H\mathbf{y}$, $\mathbf{R} = \mathbf{A}^H\mathbf{A}$, and $\mathbf{c} = \mathbf{b} - \mathbf{R}\mathbf{x}$. Removing the t-th element, $t \in I$, from the support according to the rule

$$\begin{aligned} t = \arg\min_{k \in I} & \frac{1}{2}|x_k|^2 R_{k,k} + \Re\{x_k^* c_k\} - \tau w_k |x_k| \\ \text{s.t.} \quad & \frac{1}{2}|x_t|^2 R_{t,t} + \Re\{x_t^* c_t\} - \tau w_t |x_t| < 0 \end{aligned} \tag{14}$$

reduces the cost function.

Proof. To prove this proposition, we note that when removing an element x_t from the support, the solution is updated as $\tilde{\mathbf{x}} = \mathbf{x} - x_t \mathbf{e}_t$. The change of the cost function $\Delta J = J_{\mathbf{w},\tau}(\tilde{\mathbf{x}}) - J_{\mathbf{w},\tau}(\mathbf{x})$ due to this update can be written as

$$\Delta J = \frac{1}{2}|x_t|^2 R_{t,t} + \Re\{x_t^* c_t\} - \tau w_t |x_t| \tag{15}$$

With the condition that $\Delta J < 0$, from (15), the rule (14) follows. □

The homotopy DCD (we call it Hℓ_1-DCD) algorithm for solving the $\ell_2\ell_1$ minimization problem is now presented in Table III. This algorithm is similar to the algorithm presented in [11] with the difference that the algorithm in [11] does not have a mechanism for removing elements from the support. At each homotopy iteration, the Hℓ_1-DCD algorithm first minimizes the cost function on the support I for the current value of the regularization parameter τ. Then the algorithm updates the support by removing and/or adding elements from/into the support with further reduction of the cost function.

The order of first removing and then adding elements is chosen due to the following reasons. The intention here is to keep the fixed-point M_b-bit format of elements in the solution vector $\mathbf{x}$. If we first add an element into the support, we have to update the element as defined in (10), and this would change the format as there is no guarantee that the update will provide exactly an M_b-bit word. If we do the adding last, then we do not need to do the assignment (10) and we can keep the element equal to zero since afterward we solve the $\ell_2\ell_1$

minimization on the support by using DCD iterations and this element will be properly updated within the fixed-point format. When removing an element from the support, we assign to this element a zero-value which is exactly described within the M_b-bit fixed-point format. Moreover, with this order of updating the support, we avoid the division operations in (10) that are complicated for hardware implementation. Note that computations in (9) do require division by $R_{k,k}$. However, if the dictionary $\mathbf{A}$ is normalized so that $R_{k,k} = 1$ or if $1/R_{k,k}$ is precomputed, the divisions are avoided.

The regularization parameter τ is reduced at each homotopy iteration starting from the maximum value τ_{max} found at step 1 down to a predefined value τ_{min} using the update $\tau \leftarrow \gamma\tau$. The parameter $\gamma \in (0, 1)$ determines how finely we sample the regularization parameter. The closer γ to one, the more accurate the sampling is, but more computations will be required. The algorithm will terminate if $\tau \leq \tau_{\text{min}}$, where $\tau_{\text{min}} = \mu_\tau \tau_{\text{max}}$ and $\mu_\tau \in [0, 1]$ is a predefined parameter. We will also set a limit L_{max} to the number of homotopy iterations after which the algorithm terminates.

In this algorithm, we have to solve the minimization problem at every homotopy iteration as the parameter τ affects the optimization result. For this purpose we can use the ℓ_1-DCD algorithm as described above with the following modification. The optimization is now performed on the currently identified support I only (and not the entire vector $\mathbf{x}$). This significantly speeds up the computation as the support size is usually smaller than N. However, we keep updating all elements of the residual vector $\mathbf{c}$ as they are required for updating the support after the $\ell_2\ell_1$ optimization step is finished.

The complexity of the Hℓ_1-DCD algorithm in terms of real-valued flops is given by

$$\begin{aligned} P_{\text{H}\ell_1\text{-DCD}} &= 8MN + 4N + 11C_i + 2NC_u \\ &+ 4NL + 2N_{\text{deb}}L_g \end{aligned} \tag{16}$$

This is similar to the complexity of the ℓ_1-DCD algorithm given by (8), but there are two differences. The term $4NL$ is the complexity of computing square magnitudes $|c_k|^2$ of the residual vector for checking for termination of DCD iterations; now this is checked for every homotopy iteration, thus the factor L (L is the total number of homotopy iterations). The total number of DCD iterations C_i (with the complexity $11C_i$) is now significantly reduced since solving the problem in (2) in each homotopy iteration is performed only on the currently identified support that in general is significantly smaller than the problem size N. The difference in complexity will be demonstrated in Section VI.

V. Homotopy direct-LS algorithm and homotopy DCD algorithm for $\ell_2\ell_0$ optimization

Consider the minimization of the cost function

$$J_\lambda(\mathbf{x}) = \frac{1}{2}||\mathbf{y} - \mathbf{A}\mathbf{x}||_2^2 + \lambda||\mathbf{x}||_0 \tag{17}$$

This is a non-convex problem and its solution is NP-hard. For finding an approximate solution to this problem, we will use homotopy in the regularization parameter λ. If λ is high, the second term in (17) dominates the cost function and, at very high λ, the support I of the solution must be empty, i.e., $I = \emptyset$. Therefore, it is intuitive to start the homotopy iterations with a high λ and zero support. We need to find a minimum value λ_{max} of the regularization parameter λ for which still $I = \emptyset$. Starting from $\lambda = \lambda_{\text{max}}$ in the homotopy iterations, λ will be gradually reduced using the update $\lambda \leftarrow \gamma\lambda$, where $\gamma \in [0, 1)$, so that new elements can be added into the support and/or removed from the support. Thus we need to derive rules for updating the support.

For fixed λ and I, the second term of the cost function $J_\lambda(\mathbf{x})$ in (17) is also fixed as $\lambda|I|$, and minimizing $J_\lambda(\mathbf{x})$ is equivalent to minimizing the first term, i.e., solving the LS problem on the support I. We can alternate between updating the support, which affects the second term in (17), and solving the LS problem on the fixed support, which affects the first term in (17). Adopting this strategy will allow arriving at a low complexity greedy algorithm.

Proposition 3: Let I be the support at some homotopy iteration and the current solution to the $\ell_2\ell_0$ minimization be $\mathbf{x}$. Let $\mathbf{b} = \mathbf{A}^H\mathbf{y}$, $\mathbf{R} = \mathbf{A}^H\mathbf{A}$, and $\mathbf{c} = \mathbf{b} - \mathbf{R}\mathbf{x}$. Adding the t-th element, $t \in I^c$, into the support according to the rule

$$t = \arg\max_{k \in I^c} \frac{|c_k|^2}{R_{k,k}} \quad \text{s.t.} \quad |c_t|^2 > 2\lambda R_{t,t} \tag{18}$$

and assigning to x_t the value $c_t/R_{t,t}$ reduces the cost function (17). The value of the regularization parameter λ for starting the homotopy iterations is given by $\lambda_{\text{max}} = (1/2)\max_k |b_k|^2/R_{k,k}$.

Proof. (The proof is similar to the proof of Proposition 1) We notice that $x_t = 0$ and we want to assign to this element a new value $\tilde{x}_t = \alpha$, i.e., obtain a new solution $\tilde{\mathbf{x}} = \mathbf{x} + \alpha\mathbf{e}_t$, so that the cost function is reduced. Denoting $\Delta J_\lambda = J_\lambda(\tilde{\mathbf{x}}) - J_\lambda(\mathbf{x})$, we need to check if an α exists that can reduce the cost function, i.e., make $\Delta J_\lambda < 0$. We can write

$$\Delta J_\lambda = \frac{1}{2}|\alpha|^2 R_{t,t} - |\alpha|\Re\{e^{-j\arg(\alpha)}c_t\} + \lambda \tag{19}$$

The minimum of ΔJ_λ over $\arg(\alpha)$ is achieved at $\arg(\alpha) = \arg(c_t)$. The value of the minimum is given by

$$\Delta J_\lambda = \frac{1}{2}|\alpha|^2 R_{t,t} - |\alpha||c_t| + \lambda \tag{20}$$

Minimizing (20) over $|\alpha|$ we arrive at $|\alpha| = |c_t|/R_{t,t}$. Thus, ΔJ_λ is minimized at $\alpha = c_t/R_{t,t}$ and the minimum is given by

$$\Delta J_\lambda = -\frac{|c_t|^2}{2R_{t,t}} + \lambda \tag{21}$$

Rule (18) follows from the greedy strategy of adding to the support only the entry for which the cost reduction is biggest. □

This is not the optimal way of finding an element to be added into the support, however it is simple and efficient as shown below.

This rule also determines the starting value of λ. At the start of the homotopy iterations, the support is empty, i.e., $I = \emptyset$, and $\mathbf{x} = \mathbf{0}$. Therefore, the first element to be added into I is

TABLE IV

ℓ_0 HOMOTOPY ALGORITHM

	Initialization: $\mathbf{x}=\mathbf{0}$, $I=\emptyset$, $\mathbf{b}=\mathbf{A}^H\mathbf{y}$, $\mathbf{R}=\mathbf{A}^H\mathbf{A}$
1:	$t=\arg\max_k \lvert b_k\rvert^2/R_{k,k}$, $\lambda=0.5\lvert b_t\rvert^2/R_{t,t}$, $I=\{t\}$
2:	**Repeat** until a termination condition is met:
3:	**If** the support I has been updated **then**
4:	Solve $\mathbf{R}_I\mathbf{x}_I=\mathbf{f}_I$, where $\mathbf{R}_I=\mathbf{A}_I^H\mathbf{A}_I$ and $\mathbf{f}_I=\mathbf{A}_I^H\mathbf{y}$
5:	$\mathbf{c}\leftarrow\mathbf{b}-\mathbf{R}_I\mathbf{x}_I$
6:	Update the regularization parameter: $\lambda\leftarrow\gamma\lambda$
7:	Remove from I elements satisfying the rule (23) and, for every removed element, update $\mathbf{c}\leftarrow\mathbf{c}+x_t\mathbf{R}^{(t)}$ and set $x_t=0$
8:	Add into I elements satisfying the rule (18) and (in the ℓ_0 direct-LS algorithm) set $x_t=c_t/R_{t,t}$ and update $\mathbf{c}\leftarrow\mathbf{c}-x_t\mathbf{R}^{(t)}$ or (in the Hℓ_0-DCD algorithm) set $x_t=0$

TABLE V

SOLVING THE k-TH LS PROBLEM

	Input: $\mathbf{x}_{k-1}$, $\mathbf{P}_{k-1}=\mathbf{R}_{k-1}^{-1}$, $[\mathbf{r}_a;r_b]=\mathbf{R}_{I,I}^{(k)}$, $[\mathbf{b}_a;b_b]=\mathbf{b}_I$
1:	$\mathbf{z}=\mathbf{P}_{k-1}\mathbf{r}_a$
2:	$q=1/(r_b-\mathbf{r}_a^H\mathbf{z})$
3:	$v=\mathbf{z}^H\mathbf{b}_a$
4:	$x_b=q(b_b-v)$
5:	$\mathbf{x}_a=\mathbf{x}_{k-1}-x_b\mathbf{z}$
6:	Compute $\mathbf{P}_k=\mathbf{R}_k^{-1}$ using (28)
	Output: $\mathbf{x}_k=[\mathbf{x}_a,x_b]^T$ and $\mathbf{P}_k$

TABLE VI

DCD ITERATIONS FOR LS MINIMIZATION

	Input: $\mathbf{x}$, $\mathbf{c}$, I, $\mathbf{R}$
	Initialization: $s=0$, $\delta=H$
1:	**for** for $m=1,\ldots,M_b$ **do** until $s=N_u$:
2:	$\delta=\delta/2$, $\boldsymbol{\alpha}=[\delta,\ -\delta,\ j\delta,-j\delta]$, Flag = 0
3:	**for** $n=1,\ldots,\lvert I\rvert$ **do**: $p=I(n)$
4:	**for** $k=1,\ldots,4$ **do**
5:	**if** $\Re\{\alpha_k c_p^*\}>R_{p,p}\delta^2/2$ **then**
6:	$x_p\leftarrow x_p+\alpha_k$, $\mathbf{c}\leftarrow\mathbf{c}-\alpha_k\mathbf{R}^{(p)}$
7:	Flag = 1, $s\leftarrow s+1$
8:	**if** Flag = 1 **go to** step 3

defined as

$$t=\arg\max_k \frac{|b_k|^2}{R_{k,k}} \quad \text{and} \quad |b_t|^2=2\lambda_{\max}R_{t,t} \tag{22}$$

Thus, the maximum value of λ at the start of the homotopy iterations is given by $\lambda_{\max}=(1/2)\max_k|b_k|^2/R_{k,k}$. The optimal value of the first added element is $x_t=b_t/R_{t,t}$.

This rule is similar to the rule of adding new elements into the support in the MP and OMP algorithms, except that in these two algorithms the condition $|c_t|>2\lambda R_{t,t}$ is not checked, as if it had been assumed that $\lambda=0$. Besides, in the MP and OMP algorithms, elements cannot be removed from the support. We now introduce a rule for removing an element from the support, which can be important to improve performance in some situations.

Proposition 4: Let I be the support at some homotopy iteration and the current solution to the $\ell_2\ell_0$ minimization be $\mathbf{x}$. Let $\mathbf{b}=\mathbf{A}^H\mathbf{y}$, $\mathbf{R}=\mathbf{A}^H\mathbf{A}$, and $\mathbf{c}=\mathbf{b}-\mathbf{R}\mathbf{x}$. Removing the t-th element, $t\in I$, from the support according to the rule

$$t=\arg\min_{k\in I}\left[\frac{1}{2}|x_k|^2R_{k,k}+\Re\{x_k^*c_k\}\right] \quad \text{s.t.} \quad \frac{1}{2}|x_t|^2R_{t,t}+\Re\{x_t^*c_t\}<\lambda \tag{23}$$

reduces the cost function.

Proof. To prove Proposition 4, we notice that when removing the t-th element x_t from the support and making it zero, the cost function changes by the value

$$\Delta J_\lambda=\frac{1}{2}|x_t|^2R_{t,t}+\Re\{x_t^*c_t\}-\lambda \tag{24}$$

From (24), we obtain that the condition $\Delta J_\lambda<0$ and the highest decrement of the cost function is achieved for the element defined in (23). □

Note that when adding and removing elements from the support, several elements can satisfy the conditions in (18) and (23). Thus, several elements can be simultaneously added and/or removed from the support.

The general structure of the proposed algorithm for solving the $\ell_2\ell_0$ minimization problem is now presented in Table IV. For comparison purposes, we present two options for solving the least-squares problems that appear in Table IV: recursive matrix inversion, as described in Table V, and DCD iterations, as described in Table VI. As our simulations show, the performance of the two algorithms is similar, but the complexity of the latter is much smaller.

The complexity of the ℓ_0 homotopy algorithm in terms of real-valued operations is given by

$$\begin{aligned} P_{\ell_2\ell_0} &\approx 8MN+4N+P_{\text{LS}}+4NL_g(L_g+1) \\ &+ 12NL_g+4L_g^3 \end{aligned} \tag{25}$$

The first and second terms in (25) are the complexity of computing the vector $\mathbf{b}$ and selecting the first element into the support at step 1 in Table IV. The term P_{LS} is for solving the LS problems at step 4. The LS problem can be solved using a direct approach that would result in a complexity of $\mathcal{O}(L^4)$. This can be reduced using the Cholesky factorization or conjugate-gradient iterations [33], [66], [67]. A low complexity can also be achieved using a recursive inversion of the matrix $\mathbf{R}_{I,I}$; in this case, we have $P_{\text{LS}}\approx 4L_g^3$, where L_g denotes the cardinality of the finally identified support. Details of the LS recursion are given in Appendix and the recursion is summarized in Table V; here, it is assumed that $\mathbf{R}_k=\mathbf{R}_{I,I}$. The term $4NL_g(L_g+1)$ is for updating the residual $\mathbf{c}$ at step 5 in Table IV. Adding new elements into the support (step 8) has the complexity $12NL_g$. The last term in (25) is the complexity of the debiasing after the support is identified. Removing elements from the support does not involve significant computations and it is not included in $P_{\ell_2\ell_0}$.

Similarly with the Hℓ_1-DCD algorithm, the ℓ_0 homotopy algorithms have a combination of stopping criteria. The algorithms will terminate if $\lambda\le\lambda_{\min}$, where $\lambda_{\min}=\mu_\lambda\lambda_{\max}$ and $\mu_\lambda\in[0,1]$ is a predefined parameter. We will also set a limit $L_{\max}$ to the number of homotopy iterations after which the algorithm terminates.

For a computationally efficient solution for the sequence of LS problems in the ℓ_0 homotopy algorithm we can use DCD iterations as shown in Table VI. The algorithm in Table VI should replace steps 3, 4 and 5 in Table IV. With a limited

number of DCD iterations, one cannot guarantee obtaining the exact LS solution at each homotopy iteration. Therefore, the DCD iterations will be used even when the support is not updated, i.e., step 4 and 5 in Table IV are performed in all homotopy iterations. See more discussion on this matter in [67]. When the DCD iterations start, the vectors $\mathbf{x}$ and $\mathbf{c}$ are not zeros; here, a *warm start* of the iterations is used. It is beneficial to use the LS solution found at the previous homotopy iteration as initialization of the DCD algorithm at the current homotopy iteration. As a result, a few DCD iterations may be enough to obtain an accurate LS solution at each homotopy iteration. Elements of $\mathbf{x}$ are only updated within the support, whereas all elements of $\mathbf{c}$ are updated. This is necessary for the support update stage of the ℓ_0 homotopy to decide on new elements to be added into the support. The algorithm requires a parameter N_u defining the maximum number of *successful* DCD iterations, i.e., iterations where the solution is updated. This is necessary for controlling the complexity since the main contribution to the complexity is due to the *successful* iterations.

The complexity of the resulting Hl_0-DCD algorithm, performing the ℓ_0 homotopy and using DCD iterations for solving the LS problems, is given by

$$\begin{aligned} P_{\text{H}l_0\text{-DCD}} &\approx 8MN + 4N + 2C_uN + C_i \\ &+ 4NL_g + 2N_{\text{deb}}L_g \end{aligned} \tag{26}$$

The first and second terms are for computation of the vector $\mathbf{b}$ and selection of the first element in the support at step 1 in Table IV. The term $2C_uN$ is for updating $\mathbf{c}$ in the C_u *successful* DCD iterations (step 6 in Table VI). The term C_i takes into account C_i (total number of DCD iterations) tests at step 5 of Table VI to decide if the DCD iteration is successful. Adding new elements into the support (step 8 in Table IV) has a complexity given by the term $4NL_g$. Note that, in the ℓ_0-DCD algorithm, we do not set $x_t = c_t/R_{t,t}$ to keep the fixed-point representation of the solution, but instead set x_t to zero; thus, the complexity of this step is reduced comparing to the case of the ℓ_0 direct-LS algorithm. The debiasing (the last term) is now done by using extra N_{deb} DCD iterations at the finally identified support. Note that, in the ℓ_0-DCD algorithm, solving the LS problems, which is the most computationally demanding part of the algorithm, is performed using only addition operations. This makes the ℓ_0-DCD algorithm very attractive for hardware design, e.g. on FPGA.

VI. Numerical results

In this section, we compare the MSE performance and complexity of the proposed ℓ_1-DCD, Hℓ_1-DCD, direct-LS ℓ_0-homotopy, and Hℓ_0-DCD algorithms with the MP and YALL1 algorithms.

In the YALL1 algorithm, we adjust parameters to guarantee the highest performance and low complexity. The complexity of the YALL1 algorithm is counted based on the number of matrix-vector products, assuming that these are general-type products without using fast transforms, i.e., every such product involves $8MN$ real-valued operations; when implementing algorithms on FPGAs, it is sometimes preferable to avoid

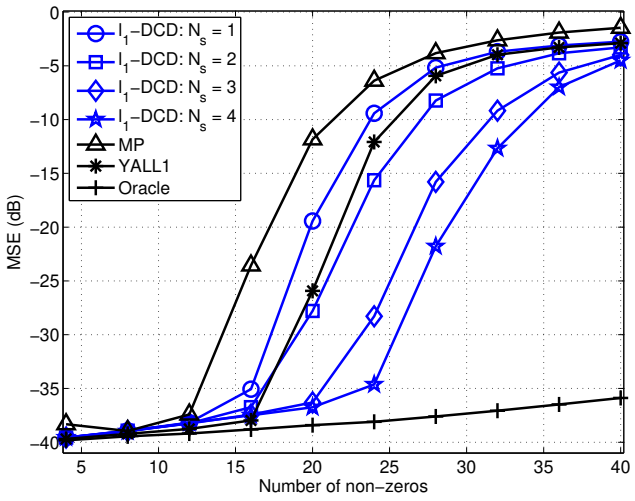

Fig. 1. MSE performance of the ℓ_1-DCD algorithm. Parameters of the scenario: $M = 64$, $N = 256$, $\sigma = 0.01$. Parameters of the algorithms: $L_{\max} = 80$, $\mu_d = 0.02$, $\mu_c = 0.02$, $\mu_\tau = 0.02$, $\gamma = 0.9$, $\beta = 0.5$, $T_w = 1$, $\mu_w = 0.5$, $M_b = 8$, $H = 4$, $N_u = 4096$.

using FFTs, but instead directly compute the products [22]. However, we will also present an example when the FFT can be used. The MP algorithm terminates on achieving the maximum residual magnitude equal to $\mu_c \max_k |b_k|$, where $\mu_c \in [0, 1]$, or a maximum number of greedy iterations $L_{\max}$. For both the YALL1 and MP algorithms, the threshold for debiasing is set to $\mu_{\text{d}} = 0.02$ [see equation (4)]. On all the plots below, we will also show the MSE performance of an *oracle* algorithm that, in each simulation trial, performs the debiasing on the true support.

We consider simulation scenarios corresponding to channel estimation in communication systems. The channel output is given by $\mathbf{y} = \mathbf{A}\mathbf{x}_0 + \mathbf{n}$, where $\mathbf{A}$ is a matrix defined by pilot symbols and $\mathbf{x}_0$ is the channel impulse response. The matrix $\mathbf{A}$ is an $M \times N$ circulant matrix generated from an $N \times 1$ vector of pilot symbols; for more details on generating the matrix $\mathbf{A}$ in channel estimation scenarios see e.g. [1], [4], [7]. The pilot symbols are independent zero-mean Gaussian random numbers. In each simulation trial, a new pilot signal, new channel impulse response $\mathbf{x}_0$, and new realization of noise are generated. The positions of the K non-zero elements in $\mathbf{x}_0$ are chosen randomly, the non-zero elements are generated as independent complex-valued Gaussian zero-mean random numbers of unit variance and then $\mathbf{x}_0$ is normalized to energy K. The noise vector $\mathbf{n}$ contains complex-valued random Gaussian entries of variance σ^2. For a fixed K, we run 1000 simulation trials and average the MSE

$$\text{MSE} = \frac{||\mathbf{x} - \mathbf{x}_0||_2^2}{||\mathbf{x}_0||_2^2}$$

obtained in the trials, where $\mathbf{x}_0$ is the true vector (channel impulse response to be estimated) and $\mathbf{x}$ is its estimate.

In the proposed DCD-based algorithms, for debiasing, we use extra $N_{\text{deb}} = 256$ DCD iterations.

We consider the case of $N = 256$ and $M = 64$. Fig. 1 and Fig. 2 show the MSE performance and complexity of the pro-

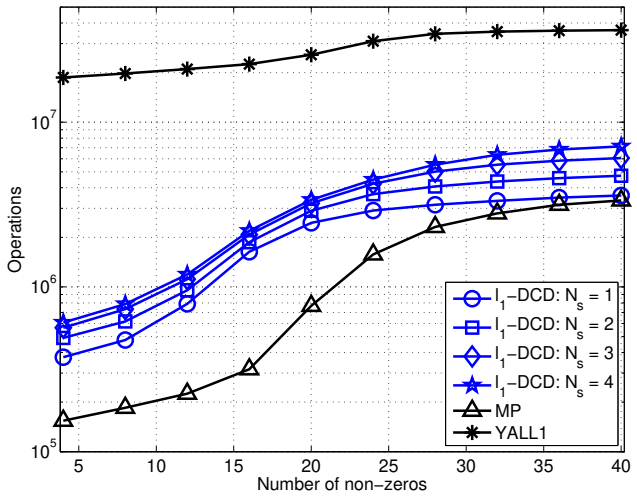

Fig. 2. Complexity of the ℓ_1-DCD algorithm. Parameters of the scenario: $M = 64$, $N = 256$, $\sigma = 0.01$. Parameters of the algorithms: $L_{\max} = 80$, $\mu_d = 0.02$, $\mu_c = 0.02$, $\mu_\tau = 0.02$, $\gamma = 0.9$, $\beta = 0.5$, $T_w = 1$, $\mu_w = 0.5$, $M_b = 8$, $H = 4$, $N_u = 4096$.

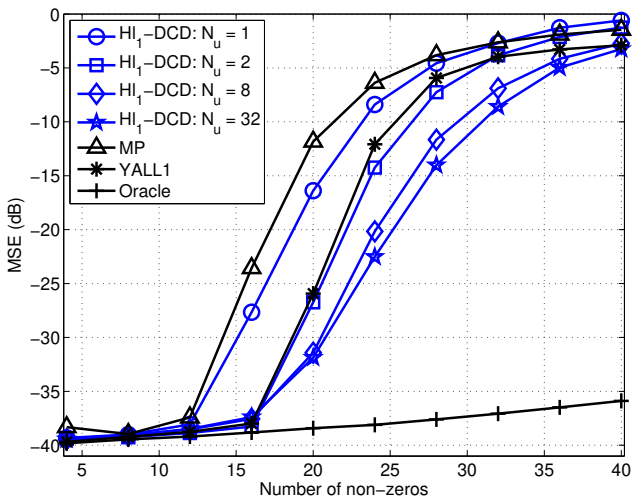

Fig. 3. MSE performance of the Hℓ_1-DCD algorithm. Parameters of the scenario: $M = 64$, $N = 256$, $\sigma = 0.01$. Parameters of the algorithms: $L_{\max} = 80$, $\mu_d = 0.02$, $\mu_c = 0.02$, $\mu_\tau = 0$, $\gamma = 0.9$, $N_s = 1$, $M_b = 8$, $H = 4$.

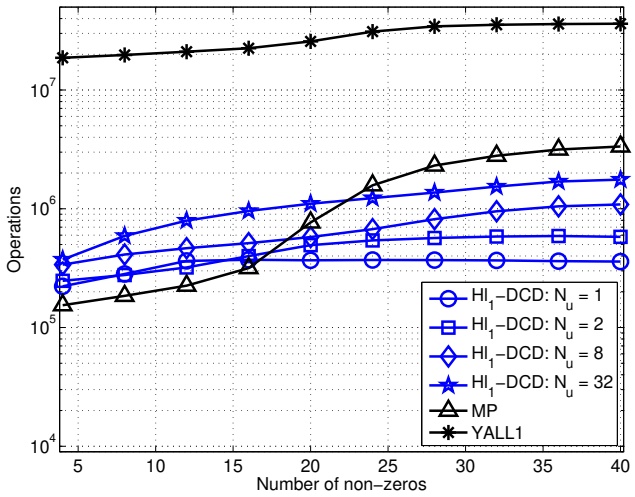

Fig. 4. Complexity of the Hℓ_1-DCD algorithm. Parameters of the scenario: $M = 64$, $N = 256$, $\sigma = 0.01$. Parameters of the algorithms: $L_{\max} = 80$, $\mu_d = 0.02$, $\mu_c = 0.02$, $\mu_\tau = 0$, $\gamma = 0.9$, $N_s = 1$, $M_b = 8$, $H = 4$.

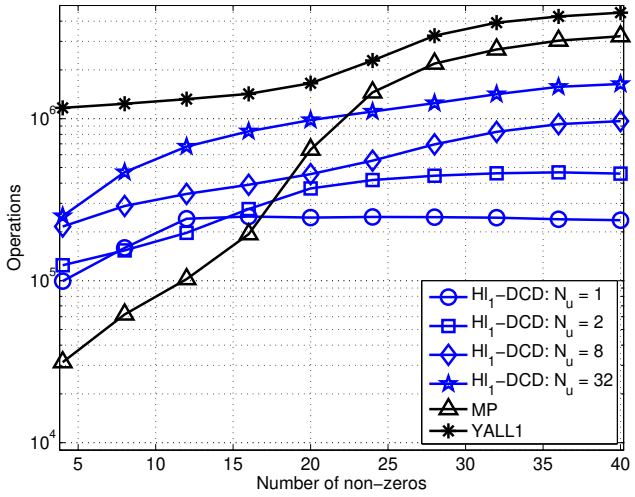

Fig. 5. Complexity of the Hℓ_1-DCD algorithm. Here, it is assumed that the FFT is used for fast computation of the matrix-vector products in all the algorithms. In the MP and Hℓ_1-DCD algorithms, there is one such a product at the initialization stage. Parameters of the scenario: $M = 64$, $N = 256$, $\sigma = 0.01$. Parameters of the algorithms: $L_{\max} = 80$, $\mu_d = 0.02$, $\mu_c = 0.02$, $\mu_\tau = 0$, $\gamma = 0.9$, $N_s = 1$, $M_b = 8$, $H = 4$.

posed ℓ_1-DCD algorithm. When solving the BPDN problem (i.e., $w_k = 1$ for $k = 1, \ldots, N$, and $N_s = 1$), the YALL1 algorithm outperforms the ℓ_1-DCD algorithm. However, the reweighting iterations allow significant improvement in the ℓ_1-DCD performance. With one weight update ($N_s = 2$), the ℓ_1-DCD performance matches the YALL1 performance. With extra reweighting iterations up to $N_s = 4$, the performance is further improved almost reaching its best at $N_s = 4$; additional iterations do not bring significant improvement. Note that the ℓ_1-DCD complexity does not increase considerably with increasing N_s (due to the *warm start* of the reweighting iterations) and it is considerably lower than the YALL1 complexity. However, the complexity of sparse recovery can be reduced when using the Hℓ_1-DCD algorithm, as shown in Fig. 4.

Fig. 3 and Fig. 4 show the MSE performance and complexity of the Hℓ_1-DCD algorithm without reweighting for different limits N_u to the number of *successful* DCD iterations. It is seen that the Hℓ_1-DCD algorithm achieves the YALL1 performance with as few as $N_u = 2$ *successful* DCD iterations per one homotopy iteration. In this case, the complexity of the Hℓ_1-DCD algorithm is close to the MP complexity or even lower. With $N_u > 2$, the Hℓ_1-DCD algorithm outperforms the YALL1 algorithm and its complexity is significantly lower than the YALL1 complexity. It is interesting to notice that the ℓ_1-DCD algorithm (which is a member of the convex optimization family as all elements of the solution vector are updated) could not achieve the YALL1 performance without the reweighting iterations, whereas the Hℓ_1-DCD algorithm (which belongs to the family of greedy algorithms) can. Fig. 5 shows complexity of the algorithms when the FFT is used for computation of the matrix-vector products. It can be seen

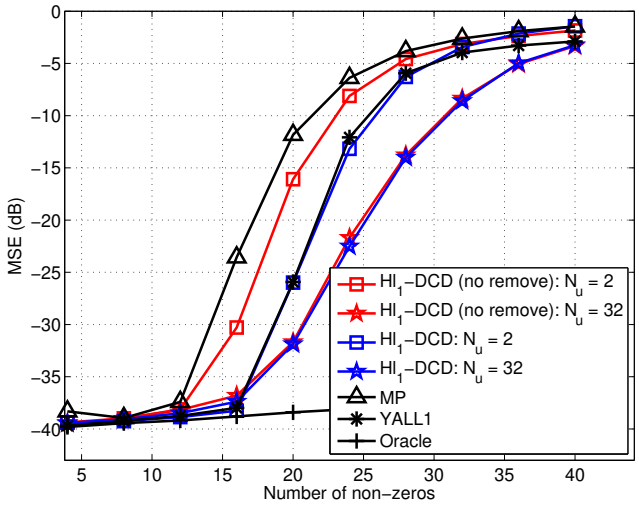

Fig. 6. MSE performance of the Hℓ_1-DCD algorithm with and without the procedure for removal of elements from the support. Parameters of the scenario: $M = 64$, $N = 256$, $\sigma = 0.01$. Parameters of the algorithms: $L_{\max} = 80$, $\mu_d = 0.02$, $\mu_c = 0.02$, $\mu_\tau = 0$, $\gamma = 0.9$, $N_s = 1$, $M_b = 8$, $H = 4$.

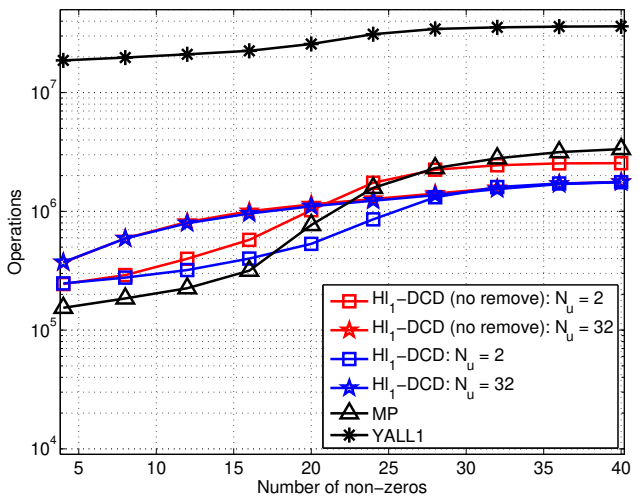

Fig. 7. Complexity of the Hℓ_1-DCD algorithm with and without the procedure for removal of elements from the support. Parameters of the scenario: $M = 64$, $N = 256$, $\sigma = 0.01$. Parameters of the algorithms: $L_{\max} = 80$, $\mu_d = 0.02$, $\mu_c = 0.02$, $\mu_\tau = 0$, $\gamma = 0.9$, $N_s = 1$, $M_b = 8$, $H = 4$.

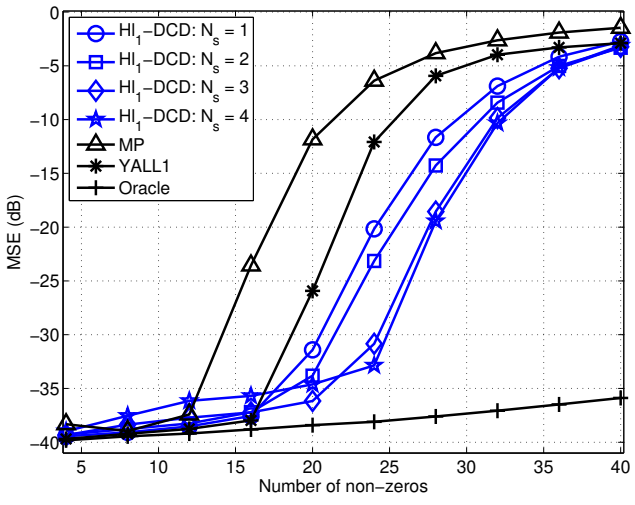

Fig. 8. MSE performance of the Hℓ_1-DCD algorithm with reweighting iterations. Parameters of the scenario: $M = 64$, $N = 256$, $\sigma = 0.01$. Parameters of the algorithms: $L_{\max} = 80$, $\mu_d = 0.02$, $\mu_c = 0.02$, $\mu_\tau = 0$, $\gamma = 0.9$, $\beta = 0.5$, $T_w = 1$, $\mu_w = 0.5$, $M_b = 8$, $H = 4$, $N_u = 8$.

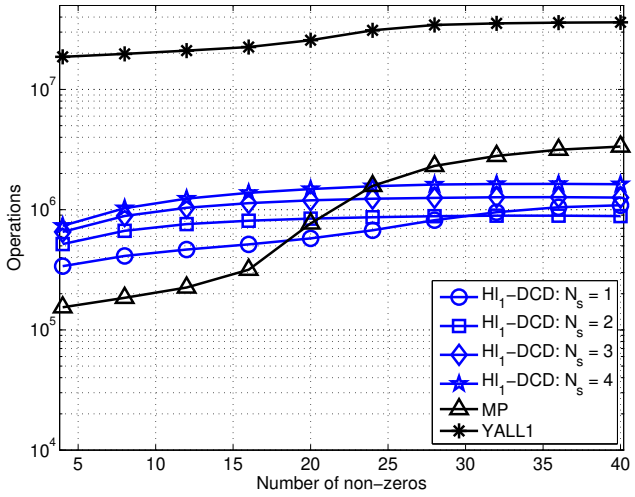

Fig. 9. Complexity of the Hℓ_1-DCD algorithm with reweighting iterations. Parameters of the scenario: $M = 64$, $N = 256$, $\sigma = 0.01$. Parameters of the algorithms: $L_{\max} = 80$, $\mu_d = 0.02$, $\mu_c = 0.02$, $\mu_\tau = 0$, $\gamma = 0.9$, $\beta = 0.5$, $T_w = 1$, $\mu_w = 0.5$, $M_b = 8$, $H = 4$, $N_u = 8$.

that the difference in complexity is now smaller, but still the Hℓ_1-DCD algorithm is significantly faster than the YALL1 algorithm.

Fig. 6 and Fig. 7 compare two versions of the homotopy algorithm. The first one is the Hℓ_1-DCD algorithm as described in Table III, and the other one is the Hℓ_1-DCD algorithm, but without removing elements from the support, i.e., without steps 5 and 6 in Table III. It is seen that, for $N_u = 2$, the effect of removing the elements is significant in both the improvement in the MSE performance and reducing the complexity. This can be explained by the fact that, with a small number of DCD iterations, due to inaccurate solving the $\ell_2\ell_1$ problem in every homotopy iteration, there are wrong elements added into the support, which, when removed, improve the performance. With the larger N_u ($N_u = 32$ in this case), the $\ell_2\ell_1$ problem is accurately solved at every homotopy iterations and the removal is not that necessary; even if some wrong elements are added into the support, the large number of DCD iterations would drive them to zero and they are removed at the hard thresholding of the debiasing stage.

The reweighting iterations (see Fig. 8 and Fig. 9) result in further improvement of the MSE performance of the Hℓ_1-DCD algorithm and the final performance is close to that achieved by the ℓ_1-DCD algorithm (compare with Fig. 1). Notice that with increase in N_s beyond $N_s = 3$, the Hℓ_1-DCD MSE curve departs from the oracle MSE curve. This can be explained by the less reliable support detection with higher N_s as the threshold for support detection is reduced and thus wrong elements are picked up in the support. Comparing Fig. 9 with

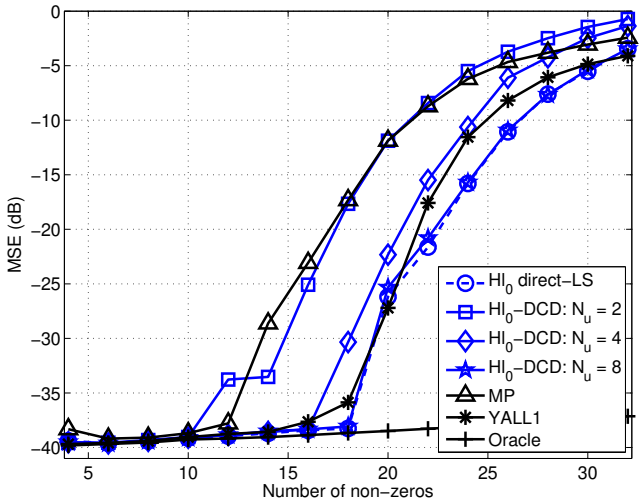

Fig. 10. MSE performance of the ℓ_0 direct-LS and Hℓ_0-DCD algorithms. Parameters of the scenario: $M = 64$, $N = 256$, $\sigma = 0.01$. Parameters of the algorithms: $L_{\max} = 80$, $\mu_d = 0.02$, $\mu_c = 0.02$, $\mu_\lambda = 0$, $\gamma = 0.9$, $M_b = 8$, $H = 4$.

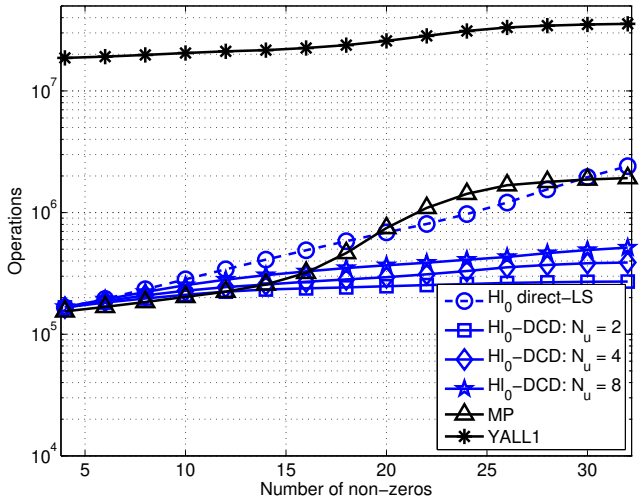

Fig. 11. Complexity of the ℓ_0 direct-LS and Hℓ_0-DCD algorithms. Parameters of the scenario: $M = 64$, $N = 256$, $\sigma = 0.01$. Parameters of the algorithms: $L_{\max} = 80$, $\mu_d = 0.02$, $\mu_c = 0.02$, $\mu_\lambda = 0$, $\gamma = 0.9$, $M_b = 8$, $H = 4$.

Fig. 2, it is seen that the complexity of the Hℓ_1-DCD algorithm is significantly lower than that of the ℓ_1-DCD algorithm.

Fig. 10 and Fig. 11 show MSE performance and complexity of the Hℓ_0 direct-LS and Hℓ_0-DCD algorithms. The Hℓ_0 direct-LS algorithm shows an MSE performance similar to that of the YALL1 algorithm. The Hℓ_0-DCD algorithm matches the YALL1 performance and the Hℓ_0 direct-LS performance with $N_u = 8$, i.e., with at most 8 DCD iterations per one homotopy iteration. The complexity of the Hℓ_0-DCD algorithm is lower than complexities of the other proposed algorithms. It is comparable to the MP complexity or even lower. Note that the LS optimization in the Hℓ_0-DCD algorithm and debiasing are multiplication-free. Thus the number of multiplications is only a small part of the whole complexity. E.g., for $N_u = 8$, only about 20% of operations are multiplications. Thus, this algorithm is well suited to hardware implementation, e.g. on FPGAs.

VII. Conclusions

In this paper, we have proposed a family of computationally efficient algorithms for recovery of complex-valued sparse signals. The algorithms are based on solving either the $\ell_2\ell_1$ or $\ell_2\ell_0$ optimization problem using dichotomous coordinate descent (DCD) iterations. We have first derived an algorithm (ℓ_1-DCD algorithm) for solving the $\ell_2\ell_1$ optimization problem with a fixed ℓ_1-regularization term. This algorithm has shown a high recovery performance and relatively low complexity; specifically, its performance is better and the complexity is lower than that of the YALL1 algorithm. The complexity has been further reduced when combining this algorithm with homotopy with respect to the regularization term. The combined algorithm (Hℓ_1-DCD algorithm) has demonstrated a performance similar to that of the ℓ_1-DCD algorithm. We have then derived an algorithm (direct-LS homotopy algorithm) for solving the $\ell_2\ell_0$ optimization problem using homotopy with respect to the ℓ_0 regularization term. Although the recovery performance of the algorithm is somewhat inferior to that of the $\ell_2\ell_1$ based algorithms, its complexity is lower and comparable to that of the matching pursuit (MP) algorithm. We have then incorporated DCD iterations into the direct-LS ℓ_0 homotopy algorithm and arrived at another algorithm (Hℓ_0-DCD algorithm) that has especially low complexity that is comparable or even lower that that of the MP complexity. Moreover, most operations required for implementation of the Hℓ_0-DCD algorithm are additions, which makes it very attractive for real-time implementation, e.g. on FPGAs.

Appendix I
Recursive Solution of the Sequence of LS Problems

Let at $(k-1)$-th iteration a system of equations $\mathbf{R}_{k-1}\mathbf{x}_{k-1} = \mathbf{b}_a$ be solved and the solution be $\mathbf{x}_{k-1} = \mathbf{R}_{k-1}^{-1}\mathbf{b}_a$, where $\mathbf{R}_{k-1} \in \mathbb{C}^{(k-1)\times(k-1)}$ and $\mathbf{x}_{k-1}, \mathbf{b}_a \in \mathbb{C}^{(k-1)\times 1}$. At the k-th iteration, we need to solve an augmented system $\mathbf{R}_k\mathbf{x}_k = \mathbf{b}$, where

$$\mathbf{R}_k = \begin{bmatrix} \mathbf{R}_{k-1} & \mathbf{r}_a \\ \mathbf{r}_a^H & r_b \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} \mathbf{b}_a \\ b_b \end{bmatrix} \tag{27}$$

The solution can be found using the formula for inversion of a block matrix as follows:

$$\mathbf{x}_k = \mathbf{R}_k^{-1}\mathbf{b} = \begin{bmatrix} \mathbf{R}_{k-1}^{-1} + q\mathbf{z}\mathbf{z}^H & -q\mathbf{z} \\ -q\mathbf{z}^H & q \end{bmatrix} \begin{bmatrix} \mathbf{b}_a \\ b_b \end{bmatrix} \tag{28}$$

where $\mathbf{z} = \mathbf{R}_{k-1}^{-1}\mathbf{r}_a$ and $q = 1/(r_b - \mathbf{r}_a^H\mathbf{z})$. We represent the solution in the form:

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_a \\ x_b \end{bmatrix} \tag{29}$$

Then, noticing that $\mathbf{x}_{k-1} = \mathbf{R}_{k-1}^{-1}\mathbf{b}_a$, we can write:

$$\begin{aligned} \mathbf{x}_a &= \mathbf{x}_{k-1} + q\mathbf{z}\mathbf{z}^H\mathbf{b}_a - q\mathbf{z}b_b \\ x_b &= -q\mathbf{z}^H\mathbf{b}_a + qb_b \end{aligned} \tag{30}$$

Thus, we arrive at the algorithm presented in Table V.

Complexity of the technique is mostly defined by steps 1 and 6, each of complexity $\mathcal{O}(k^2)$. The matrix-vector multiplication at step 1 involves about $8(k-1)^2$ real-valued operations, and generating $\mathbf{P}_k = \mathbf{R}_k^{-1}$ at step 6, taking into account that the matrix $\mathbf{R}_k$ is Hermitian, involves about $2(2k+1)(k-1)$ operations. Complexity of the other steps is $\mathcal{O}(k)$. Thus, the complexity of the LS solution update at the k-th iteration is approximately $6(2k-1)(k-1)$ real-valued operations. If k varies from 1 to L_g, the total complexity of the L_g updates is about $4L_g^3$ real-valued operations.

REFERENCES

[1] S. F. Cotter and B. D. Rao, "Sparse channel estimation via matching pursuit with application to equalization," *IEEE Transactions on Communications*, vol. 50, no. 3, pp. 374–377, 2002.

[2] G. Z. Karabulut and A. Yongacoglu, "Sparse channel estimation using orthogonal matching pursuit algorithm," in *IEEE 60th Vehicular Technology Conference, VTC2004-Fall*, 2004, vol. 6, pp. 3880–3884.

[3] W. Li and J. C. Preisig, "Estimation of rapidly time-varying sparse channels," *IEEE J. Oceanic Engineering*, vol. 32, no. 4, pp. 927–939, 2007.

[4] C. R. Berger, Z. Wang, J. Huang, and S. Zhou, "Application of compressive sensing to sparse channel estimation," *IEEE Communications Magazine*, vol. 48, no. 11, pp. 164–174, 2010.

[5] J. Huang, C. R. Berger, S. Zhou, and J. Huang, "Comparison of basis pursuit algorithms for sparse channel estimation in underwater acoustic OFDM," in *in Proceedings IEEE OCEANS 2010, Sydney*, 2010, pp. 1–6.

[6] A. Hormati and M. Vetterli, "Compressive sampling of multiple sparse signals having common support using finite rate of innovation principles," *IEEE Signal Processing Letters*, vol. 18, no. 5, pp. 331–334, 2011.

[7] C. Qi, X. Wang, and L. Wu, "Underwater acoustic channel estimation based on sparse recovery algorithms," *IET Signal Processing*, vol. 5, no. 8, pp. 739–747, 2011.

[8] E. Panayirci, H. Senol, M. Uysal, and H. V. Poor, "Sparse channel estimation and equalization for OFDM-based underwater cooperative systems with amplify-and-forward relaying," *IEEE Transactions on Signal Processing*, vol. 64, no. 1, pp. 214–228, 2016.

[9] D. Malioutov, M. Cetin, and A. S. Willsky, "A sparse signal reconstruction perspective for source localization with sensor arrays," *IEEE Trans. on Signal Processing*, vol. 53, no. 8, pp. 3010–3022, 2005.

[10] G. F. Edelmann and C. F. Gaumond, "Beamforming using compressive sensing," *J. Acoust. Soc. Am.*, vol. 130, no. 4, pp. EL232–EL237, 2011.

[11] C. Liu, Y. V. Zakharov, and T. Chen, "Broadband underwater localization of multiple sources using basis pursuit de-noising," *IEEE Transactions on Signal Processing*, vol. 60, no. 4, pp. 1708–1717, April 2012.

[12] F. G. Almeida Neto, R. de Lamare, V. Nascimento, and Y. Zakharov, "Adaptive reweighting homotopy algorithms applied to beamforming," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 51, no. 3, pp. 1902–1915, 2015.

[13] S. Cotter and B. Rao, "The adaptive matching pursuit algorithm for estimation and equalization of sparse time-varying channels," in *Proc. 34th Asilomar Conf. Signals Syst. Comput.*, 2000, vol. 2, pp. 1772–1776.

[14] D. Angelosante, J. A. Bazerque, and G. B. Giannakis, "Online adaptive estimation of sparse signals: Where RLS meets the l_1-norm," *IEEE Transactions on Signal Processing*, vol. 58, no. 7, pp. 3436–3447, 2010.

[15] E. M. Eksioglu and A. K. Tanc, "RLS algorithm with convex regularization," *IEEE Signal Processing Letters*, vol. 18, no. 8, pp. 470–473, 2011.

[16] Y. Zakharov and V. Nascimento, "DCD-RLS adaptive filters with penalties for sparse identification," *IEEE Transactions on Signal Processing*, vol. 61, no. 12, pp. 3198–3213, 2013.

[17] J. Liu and S. L. Grant, "Proportionate adaptive filtering for block-sparse system identification," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 4, pp. 623–630, 2016.

[18] Y. Meng, A. Brown, R. Iltis, T. Sherwood, H. Lee, and R. Kastner, "MP core: algorithm and design techniques for efficient channel estimation in wireless applications," in *Proc. 42nd Design Automation Conf.*, June 2005, pp. 297–302.

[19] Y. Meng, W. Gong, R. Kastner, and T. Sherwood, "Algorithm/architecture co-exploration for designing energy efficient wireless channel estimator," *ASP J. of Low Power Electronics*, vol. 1, no. 3, pp. 1–11, 2005.

[20] B. Benson, A. Irturk, J. Cho, and R. Kastner, "Survey of hardware platforms for an energy efficient implementation of matching pursuits algorithm for shallow water networks," in *Proc. 3rd ASM Int. Workshop on Underwater Networks*, 2008, pp. 83–86.

[21] J. Lu, H. Zhang, and H. Meng, "Novel hardware architecture of sparse recovery based on FPGAs," in *2nd IEEE Int. Conf. on Signal Processing Systems (ICSPS)*, 2010, pp. V1–302–V1–306.

[22] P. Maechler, P. Greisen, N. Felber, and A. Burg, "Matching Pursuit: Evaluation and implementation for LTE channel estimation," in *Proceedings of IEEE Int. Symp. Circuits and Systems (ISCAS)*, 2010, pp. 589–592.

[23] F. Ren, R. Dorrace, W. Xu, and D. Marković, "A single-precision compressive sensing signal reconstruction engine on FPGAs," in *23rd IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2013, pp. 1–4.

[24] Y. Quan, Y. Li, X. Gao, and M. Xing, "FPGA implementation of real-time compressive sensing with partial Fourier dictionary," *International Journal of Antennas and Propagation*, vol. 2016, 2016.

[25] Z. Yu, J. Su, F. Yang, Y. Su, X. Zeng, D. Zhou, and W. Shi, "Fast compressive sensing reconstruction algorithm on FPGA using Orthogonal Matching Pursuit," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2016, pp. 249–252.

[26] F. Huang, J. Tao, Y. Xiang, P. Liu, L. Dong, and L. Wang, "Parallel compressive sampling matching pursuit algorithm for compressed sensing signal reconstruction with OpenCL," *Elsevier Journal of Systems Architecture*, vol. 72, pp. 51–60, 2017.

[27] D. Yang, H. Li, G. D. Peterson, and A. Fathy, "Compressed sensing based UWB receiver: Hardware compressing and FPGA reconstruction," in *43rd Annual Conference on Information Sciences and Systems, CISS 2009*, 2009, pp. 198–201.

[28] S. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3397–3415, Dec. 1993.

[29] J. A. Tropp, "Greed is good: Algorithmic results for sparse approximation," *IEEE Trans. Inf. Theory*, vol. 50, no. 10, pp. 2231–2242, 2004.

[30] J. A. Tropp and A. C. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Transactions on Information Theory*, vol. 53, no. 12, pp. 4655–4666, 2007.

[31] Y. Zhang, "User's guide for YALL1: Your algorithms for ℓ_1 optimization," *downloaded at http://www.caam.rice.edu/optimization/*, July 2012.

[32] Z. Yang and C. Zhang, "Sparsity-undersampling tradeoff of compressed sensing in the complex domain," in *Proceedings IEEE Int. Conf. Acoustic, Speech, and Signal Processing, ICASSP*, 2011, pp. 3668–3671.

[33] S. J. Wright, R. D. Nowak, and M. A. T. Figueiredo, "Sparse reconstruction by separable approximation," *IEEE Transactions on Signal Processing*, vol. 57, no. 7, pp. 2479–2493, 2009.

[34] J. J. Fuchs, "Convergence of a sparse representations algorithm applicable to real or complex data," *IEEE Journal of Selected Topics in Signal Processing*, vol. 1, no. 4, pp. 598–605, 2007.

[35] S. Yu, K. Shaharyar, and J. Ma, "Compressed sensing of complex-valued data," *Signal Processing*, vol. 92, pp. 357–362, 2012.

[36] Y. V. Zakharov and T. C. Tozer, "Multiplication-free iterative algorithm for LS problem," *Electonics Letters*, vol. 40, no. 9, pp. 567–569, 2004.

[37] J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani, "Pathwise coordinate optimization," *The Annals of Applied Statistics*, vol. 1, no. 2, pp. 302–332, 2007.

[38] T. T. Wu and K. Lange, "Coordinate descent algorithms for Lasso penalized regression," *The Annals of Applied Statistics*, vol. 2, no. 1, pp. 224–244, 2008.

[39] M. Garcia-Magarinos, R. Cao, A. Antoniadis, and W. Gonzalez-Manteiga, "Lasso logistic regression, GSoft and the cyclic coordinate descent algorithm: Application to gene expression data," *Statistical Applications in Genetics and Molecular Biology*, vol. 9, no. 1, Article 30, pp. 1–28, 2010.

[40] J. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *J. Stat. Software*, vol. 33, no. 1, pp. 1–22, 2010.

[41] Y. Zakharov, G. White, and J. Liu, "Low complexity RLS algorithms using dichotomous coordinate descent iterations," *IEEE Transactions on Signal Processing*, vol. 56, no. 7, pp. 3150–3161, July 2008.

[42] J. Liu, Y. V. Zakharov, and B. Weaver, "Architecture and FPGA design of dichotomous coordinate descent algorithms," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 56, no. 11, pp. 2425–2438, 2009.

[43] J. Liu, B. Weaver, Y. Zakharov, and G. White, " An FPGA-based MVDR beamformer using dichotomous coordinate descent iterations," *Conf. ICC'2007, Glasgow, UK*, pp. 2551–2556, 24-28 June 2007.

[44] J. Liu, Z. Quan, and Y. Zakharov, "Parallel FPGA implementation of DCD algorithm," in *Conf. DSP'2007, Cardiff, UK*, 1-4 July 2007.

[45] D. L. Donoho and Y. Tsaig, "Fast solution of ℓ_1-norm minimization problems when the solution may be sparse," *IEEE Transactions on Information Theory*, vol. 54, no. 11, pp. 4789–4812, 2008.

[46] A. Y. Yang, S. S. Sastry, A. Ganesh, and Y. Ma, "Fast ℓ_1-minimization algorithms and an application in robust face recognition: A review," in *17th IEEE International Conference on Image Processing (ICIP)*, 2010, pp. 1849–1852.

[47] D. A. Lorenz, M. E. Pfetsch, and A. M. Tillmann, "Solving Basis Pursuit: Heuristic optimality check and solver comparison," *ACM Transactions on Mathematical Software (TOMS)*, vol. 41, no. 2, pp. 8, 2015.

[48] E. Candès, M. Wakin, and S. Boyd, "Enchancing sparsity by reweighted ℓ_1 minimization," *J. Fourier Anal. Appl.*, vol. 14, no. 5, pp. 877–905, 2008.

[49] Y. Wang and W. Yin, "Sparse signal reconstruction via iterative support detection," *SIAM Journal on Imaging Sciences*, vol. 3, no. 4, pp. 462–491, 2010.

[50] D. Wipf and S. Nagarajan, "Iterative reweighted ℓ_1 and ℓ_2 methods for finding sparse solutions," *IEEE J. Selected Topics in Signal Processing*, vol. 4, no. 2, pp. 317–329, 2010.

[51] X. Xu, X. Wei, and Z. Ye, "DOA estimation based on sparse signal recovery utilizing weighted ℓ_1 norm penalty," *IEEE Signal Processing Letters*, vol. 19, no. 3, pp. 155–158, 2012.

[52] Y. Zakharov and V. H. Nascimento, "Homotopy algorithm using dichotomous coordinate descent iterations for sparse recovery," in *Conference Record of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, 2012, pp. 820–824.

[53] T. Wang, X. Lu, Z. Xi, Z. Song, H. Tong, and W. Chen, "Off-grid sparse ISAR imaging by Hlog-DCD algorithm," in *IEEE International Radar Conference (Radar)*, 2014, pp. 1–4.

[54] T. Wang, X. Lu, X. Yu, Z. Xi, and W. Chen, "A fast and accurate sparse continuous signal reconstruction by homotopy DCD with non-convex regularization," *Sensors*, vol. 14, no. 4, pp. 5929–5951, 2014.

[55] X. Lu, T. Wang, X. Yu, C. Chen, and W. Chen, "Sparse reconstruction based reweighted non-convex optimization using homotopy-DCD algorithm," in *International Radar Conference (Radar)*, 2014, pp. 1–4.

[56] Y. Zakharov and V. H. Nascimento, "Homotopy RLS-DCD adaptive filter," in *Proceedings of the Tenth International Symposium on Wireless Communication Systems (ISWCS)*, 2013, pp. 1–5.

[57] B. Babadi, N. Kalouptsidis, and V. Tarokh, "SPARLS: The sparse RLS algorithm," *IEEE Trans. Signal Processing*, vol. 58, no. 8, pp. 4013–4025, 2010.

[58] S. Chen, D. Donoho, and M. Saunders, "Atomic decomposition by basis pursuit," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 33–61, 1998.

[59] J. Meng, W. Yin, Y. Li, N. Nguyen, and Z. Han, "Compressive sensing based high resolution channel estimation for OFDM system," *IEEE Journal of Selected Topics in Signal Processing*, vol. 6, no. 1, pp. 15–25, Feb. 2012.

[60] F. P. Vasiliev, "Numerical methods for solution of extreme problems," Nauka, Moscow (in Russian), 1980.

[61] Y. Li and S. Osher, "Coordinate descent optimization for ℓ_1 minimization with application to compressed sensing; a greedy algorithm," *Inverse Problems and Imaging*, vol. 3, no. 3, pp. 487–503, 2009.

[62] M. Al-Baali, "Descent property and global convergence of the Fletcher-Reeves method with inexact line search," *IMA Journal of Numerical Analysis*, vol. 5, pp. 121–124, 1985.

[63] Z. J. Shi and J. Shen, "Convergence of nonmonotone line search method," *Journal of Computational and Applied Mathematics, Elsevier*, vol. 193, pp. 397–412, 2006.

[64] U. Meyer-Baese, *Digital signal processing with field programmable gate arrays*, Springer, 2007.

[65] F. Auger, Z. Lou, B. Feuvrie, and F. Li, "Multiplier-free divide, square root, and log algorithms," *IEEE Signal Processing Magazine*, vol. 28, no. 4, pp. 122–126, 2011.

[66] D. L. Donoho, I. Drori, Y. Tsaig, and J. L. Starck, *Sparse solution of underdetermined linear equations by stagewise orthogonal matching pursuit*, Department of Statistics, Stanford University, 2006.

[67] T. Blumensath and M. E. Davies, "Gradient pursuits," *IEEE Transactions on Signal Processing*, vol. 56, no. 6, pp. 2370–2382, 2008.